Universität Bremen
FB 3 – Informatik
Prof. Dr. Rainer Koschke
TutorIn: Euer/Eure TutorIn

Software–Projekt 1 2013

VAK 03-BA-901.02

Architekturbeschreibung

Gruppenname

Sebastian Bredehöft	sbrede@tzi.de	2751589
Alexander Konermann	konerman@tzi.de	2596673
Hannes Bruns	habruns@tzi.de	2931964
Sylvia Kamche Tague	clara@tzi.de	2476985
Christophe Stilmant	chris@tzi.de	2728350
Jens Rahjes	jrahjes@tzi.de	2693480

Abgabe: TT. Monat JJJJ — Version 1.1

Inhaltsverzeichnis

Version und Änderungsgeschichte

Die aktuelle Versionsnummer des Dokumentes sollte eindeutig und gut zu identifizieren sein, hier und optimalerweise auf dem Titelblatt.

Version	Datum	Änderungen
1.0	TT.MM.JJJJ	Dokumentvorlage als initiale Fassung kopiert
1.1	TT.MM.JJJJ	....

1 Einführung

1.1 Zweck

Entfällt in SWP-1

Was ist der Zweck dieser Architekturbeschreibung? Wer sind die LeserInnen?

1.2 Status

Entfällt in SWP-1

1.3 Definitionen, Akronyme und Abkürzungen

1.4 Referenzen

1.5 Übersicht über das Dokument

Entfällt in SWP-1

2 Globale Analyse

2.1 Einflussfaktoren

2.1.1 Organisatorische Faktoren

Einflussfaktor	Flexibilität	Veränderbarkeit	Einfluss
O1: Management			
O1.1: Time-To-Market			
Auslieferung: am 02.08.2013 um 12 Uhr	nicht flexibel - Kein Einfluss auf die Abgabe	sehr unwahrscheinlich - Der Tutor oder die Tutorin und der Dozent entschliessen sich, ob eine Abgabe verschoben werden.	Der Zeitplan muss auf die Deadline abgestimmt sein, optionale Anforderungen können nur bei ausreichend Pufferzeit erfüllt werden
O1.2: Funktionsumfang des Produkts			
Mindestanforderungen	Alle Mindestanforderungen müssen erfüllt werden. Es gibt zusätzlich auch optionale Anforderungen, die erfüllt werden können.	Mindestanforderungen könnten wegfallen.	Dies Kann positiven Einfluss auf das Zeitmanagement und die Qualität des Produktes haben.
O2: Personal			
O2.1: Anzahl Entwickler			
6 Entwickler	Bei Ausstieg von Gruppenmitgliedern kann sich mit den Tutoren über eine Senkung der Mindestanforderungen geeinigt werden	Entwickler können aus dem Projekt aussteigen.	Auswirkungen auf Time-To-Market und Mindestanforderungen.
O2.2: Erfahrung			
Erfahrung der Entwickler	nicht flexibel	Erfahrungen werden für viele Leuten im Laufe des Projekts gesammelt.	Auswirkungen auf die Struktur der Architektur.

O3: Prozesse und Werkzeuge			
O3.1: Tests			
Testen der App	nicht flexibel	nicht veränderlich	Time-To-Market, Modularisierung
O5: Entwicklungsbudget			
O5.1: Anzahl Entwickler			
6 Entwickler	Anzahl der Entwickler ist festgesetzt	Entwickler können aus dem Projekt aussteigen.	große Auswirkungen auf den Zeitplan der einzelnen Entwickler geben, die den Ausfall kompensieren müssen. Alle Mindestanforderungen können nicht erfüllt werden.

2.1.2 Technische Faktoren

Einflussfaktor	Flexibilität	Veränderbarkeit	Einfluss
T1: Hardware			
T1.4: Plattenspeicher			
Speicherplatzbedarf größer als 40 MB	Wir können entscheiden, in welchem Format Daten intern gespeichert werden.	Die Daten der Datenbank werden anwachsen.	Auswirkungen auf Dateiformate, Implementierung.
T2: Software			
T2.1: Betriebssystem			
Die App muss für das Betriebsystem Android ausgelegt sein	nicht flexibel	Möglicherweise Unterstützung neuer Versionen gefordert	Verzichten auf Features und Eigenschaften von Android Versionen über 2.3. Aufwärtskompatibilität in der Regel gewährleistet.

T2.2: Betriebssystem II			
Server-Tool muss auf Linux und Windows lauffähig sein	nicht flexibel	Keine Veränderung zu erwarten	Je nach Wahl der Programmiersprache mehr oder weniger Implementierungsaufwand (
T2.3: Benutzerschnittstelle I			
Orientierung der GUI nach ansprechendes GUI-Design	Die graphische Gestaltung ist uns überlassen	Beachtung der Android Design-Richtlinien könnte gefordert werden	Auswirkung je nach Eintreten der Forderung. Generelle Änderungen an der Darstellung können evtl. Änderungen an der Implementierung mit sich ziehen.
T2.4: Benutzerschnittstelle II			
Anpassung der GUI an Funktionsumfang	Hinzufügen von optionalen Anforderungen ist möglich	Die Senkung der Mindestanforderungen impliziert Veränderungen	Die GUI muss stets die jeweiligen implementierten Funktionen zugänglich machen
T3: Architekturtechnologie			
T3.4: Architekturstile			
Kapselung in einzelne Module	Flexibel, da keine Vorgaben zur Verfügung gestellt wurde	Bestimmte Funktionen erfordern die Umsetzung	Bestimmte Strukturierung der Software erforderlich, etwaige Überarbeitung des Programmcodes
T4: Standards			
T4.1: Datenbank			
Gebrauch spezieller Datenbank gefordert.	Es wird eine relationale Datenbank gefordert.	Aus Sicherheits-/Kompatibilitätsgrunden wäre es klug spezielle Datenbanktechnologie zu verwenden .	Grad der Auswirkung abhängig davon, wie gut der Programmierer mit der geforderten Datenbank vertraut ist. Evtl. soll eine Technologie verwendet werden, die kein Entwickler bis dahin verwendet hat.

T4.2: Datenbank			
Alternativer Datentyp gefordert	xlsx-Format wurde abgesprochen	Aus betriebsinternen Gründen, z.B. Umstellung der Software, soll eine alternativer Datentyp verwendet werden.	Mittlere Auswirkungen auf die Datenkonvertierung

2.1.3 Produktfaktoren

Einflussfaktor	**Flexibilität**	**Veränderbarkeit**	**Einfluss**
P1: Benutzerschnittstelle			
P1.1: Benutzbarkeit			
Benutzbarkeit	Freie Hand bei der Gestaltung der Benutzbarkeit	Kunde ist nicht zufrieden mit der Benutzbarkeit der Applikation	Mittlere bis hohe Auswirkung auf die Gestaltung der Benutzeroberfläche.
P2: Performanz			
P2.1: Ausführungszeiten			
GUI muss auf Benutzereingaben reagieren	Hohe Flexibilität, weil der Kunde nichts dazu spezifiert hat	Der Kunde fordert ein Zeitlimit für die Ausführung	Rechenintensivität der Funktionen ist eingeschränkt
P2.2: Ausführungszeiten			
möglichst schnell Datenabgleich mit dem Datenbank	Hohe Flexibilität, weil der Kunde nichts dazu spezifiert hat	Der Kunde fordert ein Zeitlimit für die Ausführung	Dies wirkt auf die Implementation von Update-Funktion der App
P3: Verlässlichkeit			
P3.1: Verfügbarkeit			
App kann offline nutzbar sein	Keine Flexibilität	Der Offline-Modus kann nicht berücksichtigt werden	Nutzung der bereits heruntergeladenen und selbsterstellten Inhalte auch ohne Online-Verbindung

P3.2: Robustheit			
Kategorisierung der Begriffe durch den Benutzer möglich	Keine Flexibilität, Polyhierarschie gefordert	Keine Veränderung zu erwarten	Auswirkung auf die interne Datenbankstruktur (u.a. Vermeidung von Endlosscheifen)
P3.3: Zuverlässigkeit			
Konsistenz der Daten muss nach einem Update gewährleistet sein	Nicht flexibel	Keine Veränderung zu erwarten	die Dauer des Updates wird durch Konsistenzprüfung reduziert
P3.4: Sicherheit			
Verschlüsselung der Datenübertragung	Freie Wahl der Verschlüsselung, solange eine nicht leicht zu realisieren ist	Es könnte eine bestimmte Verschlüsselung gefordert werden	Implementierung von Ver- und Entschlüsselung
P3.5: Wartung			
Modularisierung der Bibliothek-App	Keine Flexibilität	Keine Veränderlichkeit	Auswirkung auf die Struktur der Bibliothek-App bezüglich der Implementierung

2.2 Probleme und Strategien

2.2.1 Problemkarten 001 - Arbeitsaufwand und Zeitplan

Arbeitsaufwand und Zeitplan
Unser Abgabetermin ist fest und der Arbeitsaufwand ist groß. **Einflussfaktoren** O1.1: Time-To-Market O1.2: Mindestanforderungen O2.1: Anzahl Entwickler O2.2: Erfahrung Entwickler O3.1: Tests T3.1: Architekturstile
Lösung **Strategie: Drei-Schichten-Architekturen** Da wir wenig Zeit haben, implementieren wir den Server und die App in der bereits bekannten Drei-Schichten-Architekturen. Die drei Schichten sind GUI, Logik und Daten.

2.2.2 Problemkarten 002 - Netzwerkanpassung

Netzwerkanpassung
Neue Technologien oder veränderte Hardware des Servers. Die Kommunikation funktioniert nicht im vollen Umfang oder gar nicht. Einflussfaktoren O1.1: Time-To-Market O1.2: Mindestanforderungen O2.1: Anzahl Entwickler O2.2: Erfahrung Entwickler T3.1: Architekturstile P4.3: Zuverlässigkeit P4.4: Sicherheit
Lösung **Strategie: Eigene Komponente** Wir kapseln das Netzwerkmodul des Servers und der App von der restlichen Implementierung ab, sodass Änderungen leichter vorgenommen werden können.

2.2.3 Problemkarten 003 - Netzwerkkommunikation

Netzwerkkommunikation
Die App benötigt eine Schnittstelle zum aufrufen von Funktonen auf dem Server und umgekehrt. Einussfaktoren O1.1: Time-To-Market O1.2: Mindestanforderungen O2.1: Anzahl Entwickler O2.2: Erfahrung Entwickler P3.1: Ausführungszeiten P3.2: Ausführungszeiten T2.1: Betriebssysteme T3.1: Architekturstile P4.1: Verfügbarkeit P4.3: Zuverlässigkeit P4.4: Sicherheit
Lösung **Strategie: Parser** Wir schreiben einen Parser der die Befehle jeweils in beiden Richtungen vereinheitlicht.

2.2.4 Problemkarten 004 - Mehrere Clients an einem Server

Mehrere Clients an einem Server
Es sind mehrere Clients an dem Server verbunden, der Server soll die Daten nicht an alle Clients senden. Einflussfaktoren O1.1: Time-To-Market O1.2: Mindestanforderungen O2.1: Anzahl Entwickler O2.2: Erfahrung Entwickler P3.1: Ausführungszeiten P3.2: Ausführungszeiten T2.1: Betriebssysteme P4.3: Zuverlässigkeit
Lösung **Strategie: Thread-Pool** Jeder Client bekommt einen eigenen Thread, dieser befindet sich in einem Modul das für die Request Verarbeitung zuständig ist.

2.2.5 Problemkarten 005 - Administration der Datenbank

Administration der Datenbank
Die APOLLON Redaktion muss neue Datensätze in die Datenbank des Servers einbinden können. Einflussfaktoren O1.1: Time-To-Market O1.2: Mindestanforderungen O2.1: Anzahl Entwickler O2.2: Erfahrung Entwickler T2.2: Betriebssysteme2
Lösung **Strategie: Implementieren eines Administrationstools** Wir entwickeln eine Tool, das den Datensatz des Servers ändern kann.

3 Konzeptionelle Sicht

Unser System wird in 2 große Komponenten unterteilt, in Client und Server:

Die beiden Komponenten kommunizieren über ein Netzwerk über 2 asynchrone, nachrichtenbasierte Konnektoren:

- **message**
  Bei Änderungen der Datenbank informiert der Server den Client und überträgt die

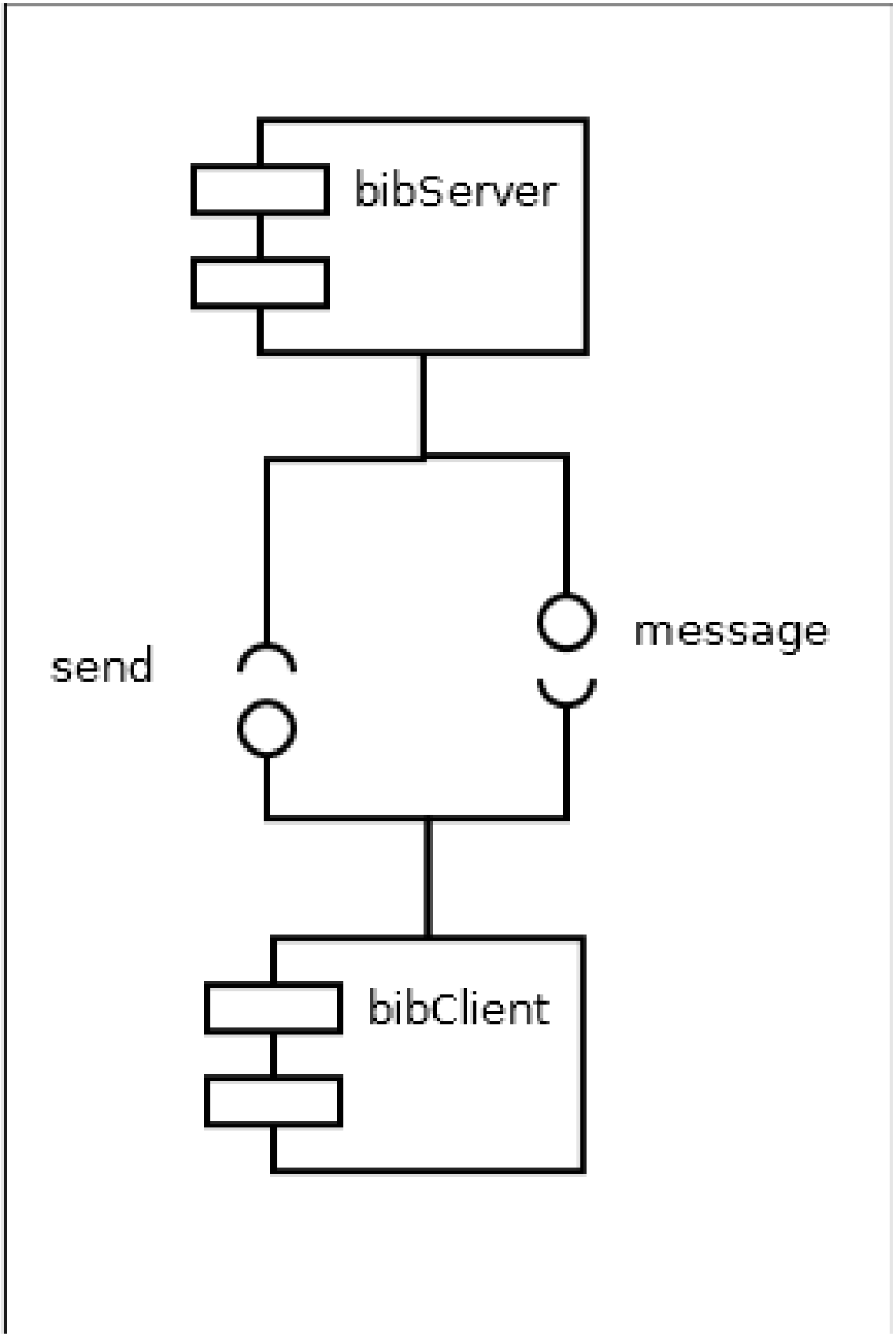


Abbildung 1: konzeptionelle Sicht

entsprechenden Daten.

- **request**
  Der Client übermittelt Anfragen an den Server. Diese Nachrichten enthalten die Information welche Methoden im Server ausgeführt werden sollen und die eventuell benötigten Daten.

Die Beiden Komponenten detailliert:

3.1 Client

Der Client, damit ist sowohl die Webseite als auch die App gemeint, besteht aus folgenden Komponenten:

- **GUI**
  Das ist die grafische Benutzeroberfläche, also der Teil mit dem der Benutzer interagiert.
- **Model** Diese Komponente bildet die Datenhaltung für die Bücher, Benutzer und Ausleihen.
  Außerdem informiert sie die GUI über Aktualisierungen des Datenbestands und nimmt gewünschte Aktionen vom Benutzer entgegen und gibt diese weiter an die Komponente Communication.
- **Communication** Communication baut eine Verbindung zum Server auf und ermöglicht so die Übertragung von Daten zwischen dem Client und Server und ist somit die Kommunikationsschnittstelle.

Die methodenbasierenden Konnektoren:

- **control**
  Die GUI übermittelt gewünschte Aktionen vom Benutzer.
- **data**
  Model übermittelt Daten die zur Darstellung benötigt werden.
- **update**
  Communication übermittelt Änderungen an den Daten.
- **send**
  Model leitet geänderte Daten bzw fordert benötigte Daten an.

3.2 Server

Der Server besteht aus folgenden Komponenten:

- **Persistence**
  Dieser Teil bildet die dauerhafte Speicherung des Datenbestands und erlaubt den Zugriff auf diese Daten.

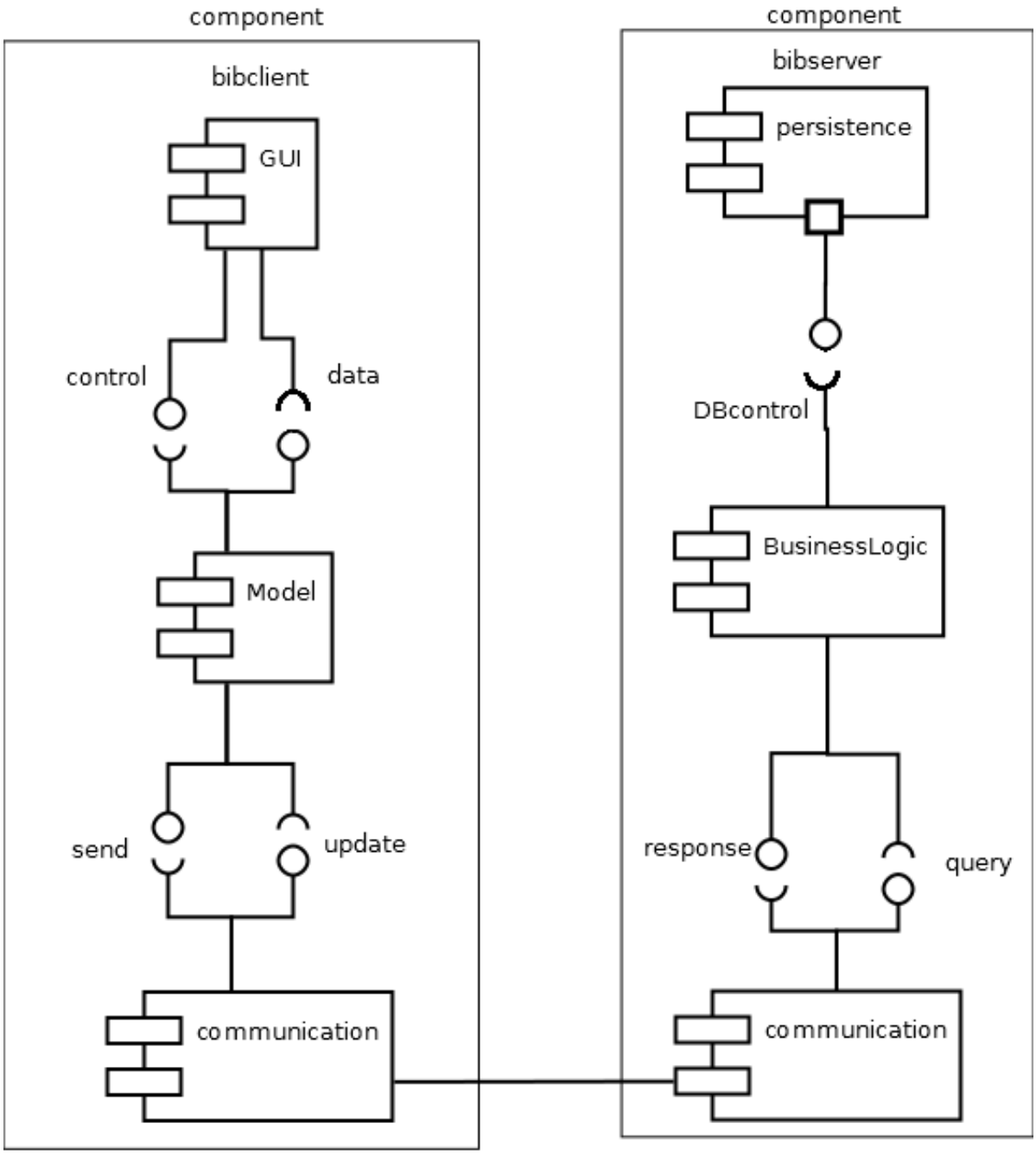


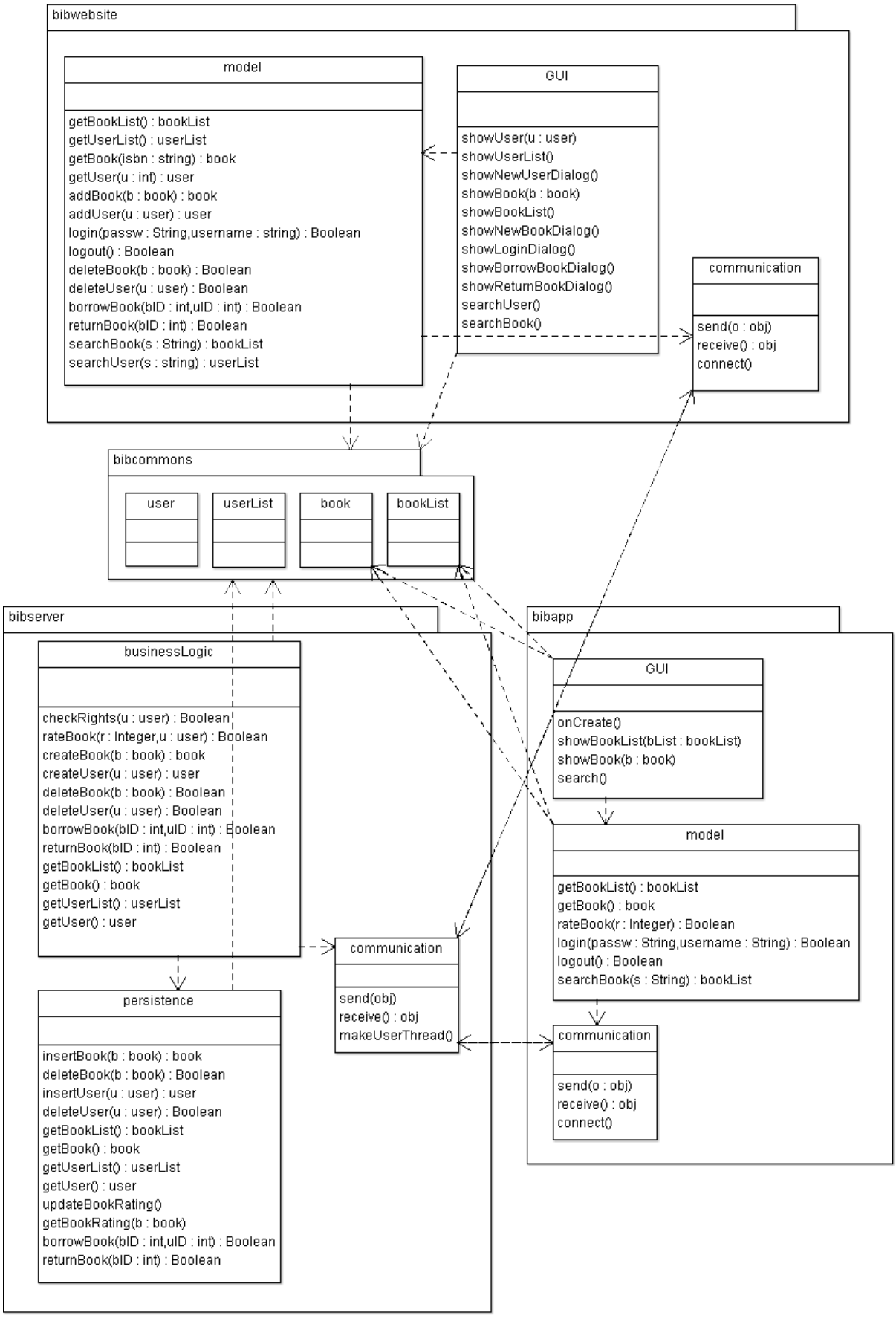
Abbildung 2: Server und Client

- **BusinessLogic**
  Die BusinessLogic realisiert die eigentliche Funktionalität. Nötige Änderungen des Datenbestands werden die Persistence weitergegeben
- **Communication**
  Communication nimmt Anfragen vom Client an leitet diese zur BusinessLogic weiter. Ebenso übermittelt sie die ihr übergegebene Daten an den Client.

Die Konnektoren:

- **DBcontrol**
  der Aufrufer übermittelt Änderungen im Datenbestand an den Aufgerufenen und übergibt diesem die Kontrolle. Dieser aktualisiert daraufhin die persistenten Daten und gibt die Kontrolle zurück.
- **query**
  Communication übermittelt der BusinessLogic nötige Änderung am Datenbestand. Die BusinessLogic übernimmt dann die Kontrolle und führt die Änderungen aus. Oder Communication stellt eine Anfrage an die BusinessLogic.
- **response**
  Die BusinessLogic übermittelt Änderungen im Datenbestand an Communication weiter und übergibt diesem die Kontrolle.

4 Modulsicht



4.0.1 bibwebsite

GUI	
showUser(u : user)	zeigt die Details eines Nutzers an
showUserList()	zeigt die Liste der Nutzer an
showNewUserDialog()	zeigt den Dialog zum hinzufügen eines Benutzers an
showBook(b : book)	zeigt die Details eines Buches an
showBookList()	zeigt die Liste der Bücher an
showNewBookDialog()	zeigt den Dialog zum Hinzufügen eines Benutzers an
showLoginDialog()	zeigt den Dialog zum Einloggen an
showBorrowBookDialog()	zeigt den Dialog zum Verleihen eines Buches an
showReturnBookDialog()	zeigt den Dialog zur Buchrückgabe an
searchBook()	suche nach Büchern
searchUser()	suche nach Nutzern
model	
getBookList() : bookList	holt die Bücherliste vom Server
getUserList() : userList	holt die Nutzerliste vom Server
getBook(isbn : string) : book	holt die kompletten Daten eines Buchs von Server
getUser(u : int) : user	holt die kompletten Daten eines Nutzers von Server
addBook(b : book) : book	fügt ein Buch zur Datenbank hinzu
addUser(u : user) : user	fügt einen Nutzer zur Datenbank hinzu
login(passw : String,username : string) : Boolean	loggt den Nutzer beim Server ein
logout() : Boolean	loggt den Nutzer beim Server aus
deleteBook(b : book) : Boolean	löscht Buch aus Datenbank
deleteUser(u : user) : Boolean	löscht Nutzer aus Datenbank
borrowBook(bID : int, uID : int) : Boolean	trägt Buch und Nutzer in die *Buch_Benutzer* Tabelle ein
returnBook(bID : int) : Boolean	trägt das Buch als zurückgegeben in die *Buch_Benutzer* Tabelle ein
searchBook(s : String) : bookList	sucht nach Büchern
searchUser(s : string) : userList	sucht nach Nutzern
communication send(o : obj)	sendet Objekte zum Server
receive() : obj	erhält Objekte vom Server
connect()	erzeugt eine Verbindung mit dem Server

4.0.2 bibserver

communication

send(obj)	sendet Objekte zum Client
receive() : obj	erhält Objekte vom Client
makeUserThread()	erzeugt einen Thread für jede eingehende Verbindung

businessLogic

checkRights(u : user) : Boolean	überprüft, ob der Client die Rechte für die angefragte Aktion besitzt
rateBook(r : Integer,u : user) : Boolean	fügt der Datenbank eine Bewertung hinzu
createBook(b : book) : book	fügt ein Buch der Datenbank hinzu
createUser(u : user) : user	fügt einen Benutzer der Datenbank hinzu
deleteBook(b : book) : Boolean	löscht ein Buch aus der Datenbank
deleteUser(u : user) : Boolean	löscht einen Nutzer aus der Datenbank
borrowBook(bID : int,uID : int) : Boolean	trägt Buch und Nutzer in die *Buch_Benutzer* Tabelle ein
returnBook(bID : int) : Boolean	trägt das Buch als zurückgegeben in die *Buch_Benutzer* Tabelle ein
getBookList() : bookList	holt die Bücherliste aus der Dantenbank
getBook() : book	holt die Details eines Buches aus der Datenbank
getUserList() : userList	holt die Liste aller Nutzer aus der Datenbank
getUser() : user	holt die Details eines Nutzers aus der Datenbank

persistence

insertBook(b : book) : book	fühgt ein Buch der Datenbank hinzu
deleteBook(b : book) : Boolean	löscht ein Buch aus der Datenbank
insertUser(u : user) : user	fügt einen Nutzer der Datenbank hinzu
deleteUser(u : user) : Boolean	löscht einen Nutzer aus der Datenbank
getBookList() : bookList	erstellt eine Liste der Bücher der Datenbank
getBook() : book	holt die Details eines Buches aus der Datenbank
getUserList() : userList	erstellt eine Liste der Nutzer der Datenbank
getUser() : user	holt die Details eines Nutzers aus der Datenbank
updateBookRating()	fügt eine neue Bewertung hinzu
getBookRating(b : book)	gibt die Bewertung eines Buches aus
borrowBook(bID : int,uID : int) : Boolean	trägt Buch und Nutzer in die *Buch_Benutzer* Tabelle ein
returnBook(bID : int) : Boolean	trägt das Buch als zurückgegeben in die *Buch_Benutzer* Tabelle ein

4.0.3 bibapp

GUI	
onCreate()	erzeugt den Startbildschirm der App
showBookList(bList : bookList)	zeigt die Liste der Bücher an
showBook(b : book)	zeigt die Details eines Buches an
search()	suche nach Büchern
model	
getBookList() : bookList	holt die Bücherliste vom Server
getBook(isbn : string) : book	holt die Details eines Buchs von Server
rateBook(r : Integer) : Boolean	bewertet Buch
searchBook(s : String) : bookList	suche nach Büchern
login(passw : String,username : String) : Boolean	loggt den Nutzer beim Server ein
logout() : Boolean	loggt den Nutzer beim Server aus
communication	
send(o : obj)	sendet Objekte zum Server
receive() : obj	erhält Objekte vom Server
connect()	erzeugt eine Verbindung mit dem Server

4.0.4 bibcommon

user
userlist
book
booklist

5 Datensicht

Das zugrundeliegende Datenmodell beschreibt die Datensicht zwischen Benutzer und Büchern in den Datenbänken.

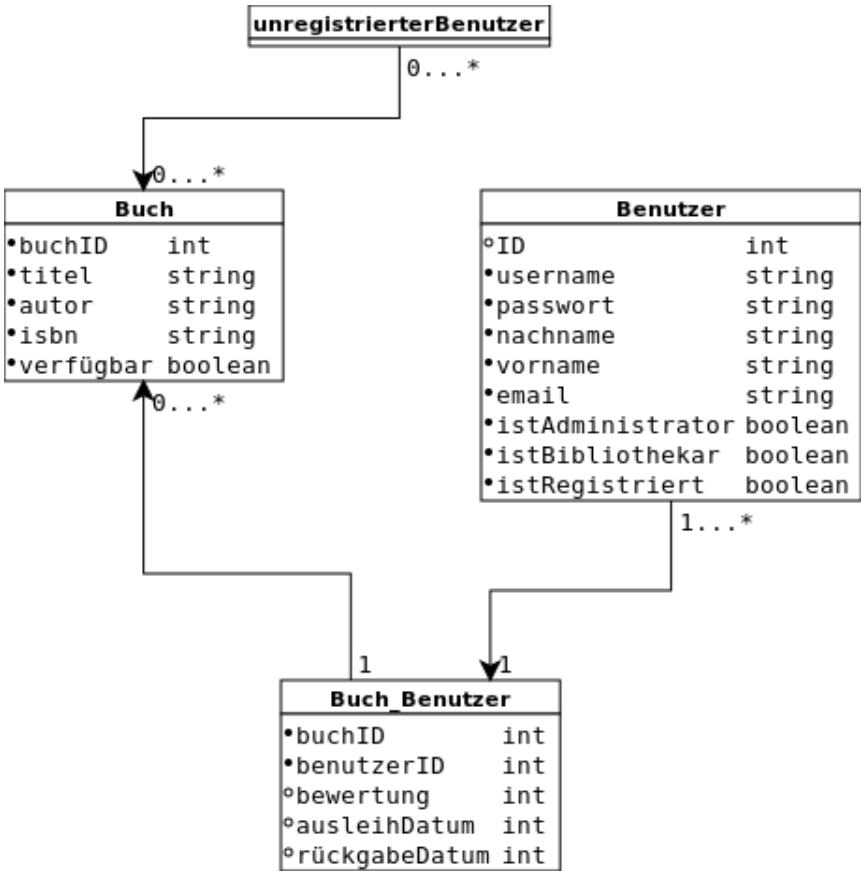


Abbildung 3: Datenmodell

Abbildung 1 zeigt das Datenmodell des Systems. Es gibt die Klasse *Benutzer*, welche die verschiedenen Nutzer des Systems repräsentiert. Jeder Nutzer hat hierbei eine feste ID die systemweit eindeutig ist.
Außerdem hat jeder einen eindeutigen Benutzernamen *username* und ein Passwort *passwort*. Diese werden bei der Registrierung ausgewählt und werden zur Anmeldung an das System benutzt.
Name, Nachname und Emailadresse werden für jeden Nutzer als String angelegt.
Desweiteren gibt es für die Klasse *Benutzer* drei booleans, welche zeigen ob ein Benutzer ein Administrator, Bibliothekar oder ein registrierter Nutzer ist. Da ein Administrator sowohl den Rang eines registrierten Benutzers als auch den eines Bibliothekar hat, ist bei diesem *istAdministrator, istBibliothekar* und *istRegistriert* auf *true* gesetzt. Bei einem Bibliothekar dementsprechend nur *istBibliothekar* und *istRegistriert* und bei einem normalen registrierten Benutzer ist nur *istRegistriert* auf *true* gesetzt.

Es gibt eine Liste (*Buch_Benutzer*) in der BuchIDs und BenutzerIDs verknüpft werden, damit zwischen Benutzer und Buch eine konkrete Verbindung besteht (damit man erkennt ob ein Benutzer ein Buch schon einmal bewertet hat, wann er es ausgeliehen hat und wann er es zurückgeben soll). Dazu stehen etwaige Bewertungen von Büchern

und das Ausleih- und Rückgabedatum.

Die Klasse *Buch* repräsentiert die Bücherliste. In dieser Liste hat jedes Buch eine eindeutige systemweite BuchID. Außerdem werden dort Titel, Autor, ISBN und die Verfügbarkeit gespeichert.

6 Zusammenhänge zwischen Anwendungsfällen und Architektur

Bei dem Diagramm wird Vorausgesetzt, dass der Benutzer als Bibliothekar eingeloggt ist und sich auf der Startseite befindet.
Beim Druck auf den "Buch hinzufügen" Button wird von der GUI ein Dialog geöffnet, in dem der Bibliothekar die Daten des Buches eintragen kann. Wenn er diese Daten mit dem "senden" Button bestätigt werden, wird ein Buchobjekt erstellt, welches über die Logik der Website, übers Netzwerk zur Logik des Servers geschickt. Hier wird sicherheitshalber geprüft ob der angemeldete Nutzer auch das Recht hat ein Buch hinzuzufügen. Ist dies der Fall wird von Persistence eine Methode aufgerufen, die das Buch in die Datenbank einträgt und die Id des Buches zurückgibt. Anschließend wird von der Logik die id dem Buch hinzugefügt. Das nun vollständige Buchobjekt wird bis zum Model zurückgegeben und der Bücherliste hinzugefügt. Die aktualisierte Liste wird anschließend von der GUI angezeigt.

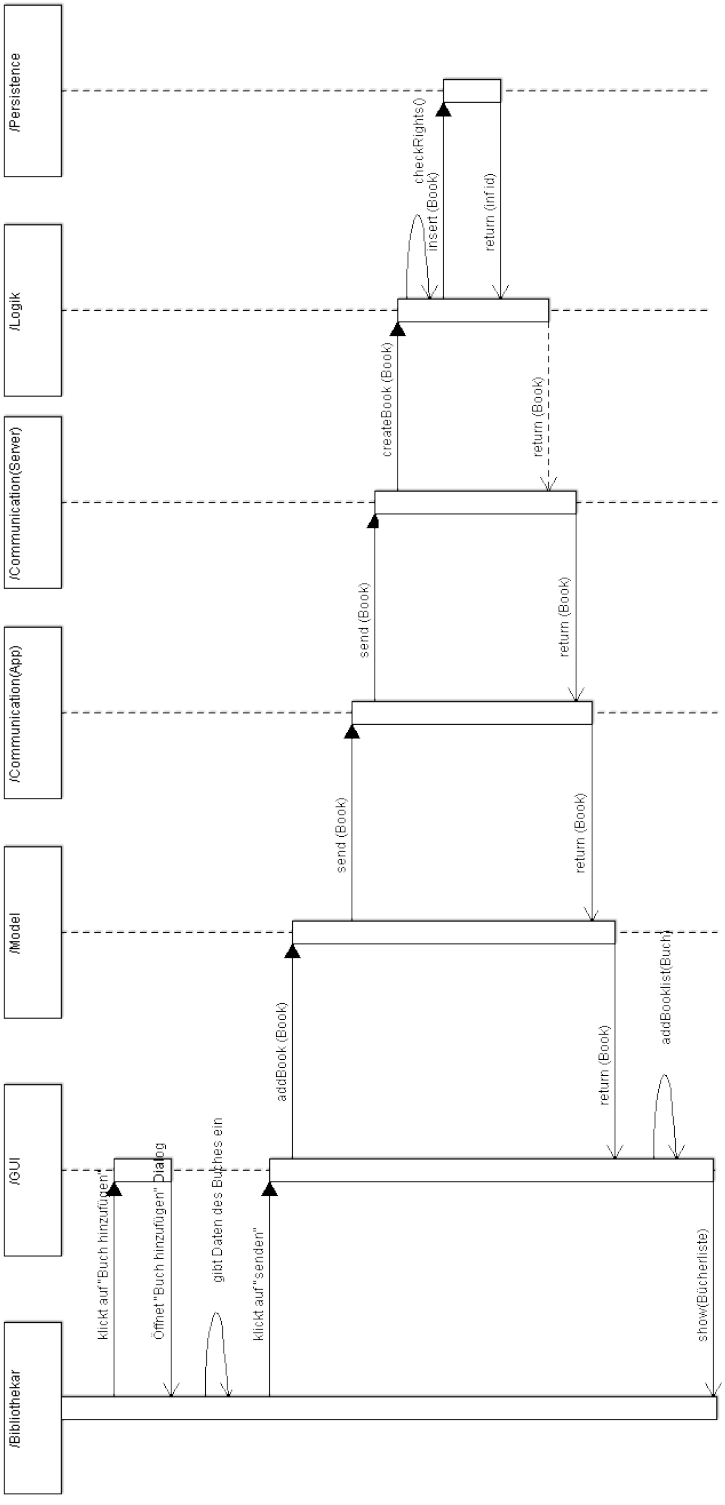


Abbildung 4: Sequenzdiagramm